OpenCore

Reference Manual (0.5.2.3)

[2019.12.02]

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/OcSupportPkg/Include
-I/UefiPackages/MacInfoPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
```

Listing 2: ECC Configuration

Warning: Tool developers modifying `config.plist` or any other OpenCore files must ensure that their tool checks for `opencore-version` NVRAM variable (see Debug Properties section below) and warn the user if the version listed is unsupported or prerelease. OpenCore configuration may change across the releases and the tool shall ensure that it carefully follows this document. Failure to do so may result in this tool to be considered as malware and blocked with all possible means.

3.4 Coding conventions

Just like any other project we have conventions that we follow during the development. All third-party contributors are highly recommended to read and follow the conventions listed below before submitting their patches. In general it is also recommended to firstly discuss the issue in Acidanthera Bugtracker before sending the patch to ensure no double work and to avoid your patch being rejected.

Organisation. The codebase is structured in multiple repositories which contain separate EDK II packages. `AppleSupportPkg` and `OpenCorePkg` are primary packages, and `EfiPkg`, `OcSupportPkg`, `MacInfoPkg.dsc`) are dependent packages.

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.
- Each unique commit should compile with `XCODE5` and preferably with other toolchains. In the majority of the cases it can be checked by accessing the CI interface. Ensuring that static analysis finds no warnings is preferred.
- External pull requests and tagged commits must be validated. That said, commits in master may build but may not necessarily work.
- Internal branches should be named as follows: `author-name-date`, e.g. `vit9696-ballooning-20191026`.
- Commit messages should be prefixed with the primary module (e.g. library or code module) the changes were made in. For example, `OcGuardLib: Add OC_ALIGNED macro`. For non-library changes `Docs` or `Build` prefixes are used.

Design. The codebase is written in a subset of freestanding C11 (C17) supported by most modern toolchains used by EDK II. Applying common software development practices or requesting clarification is recommended if any particular

case is not discussed below.

- Never rely on undefined behaviour and try to avoid implementation defined behaviour unless explicitly covered below (feel free to create an issue when a relevant case is not present).
- Use `OcGuardLib` to ensure safe integral arithmetics avoiding overflows. Unsigned wraparound should be relied on with care and reduced to the necessary amount.
- Check pointers for correct alignment with `OcGuardLib` and do not rely on the architecture being able to dereference unaligned pointers.
- Use flexible array members instead of zero-length or one-length arrays where necessary.
- Use static assertions (`STATIC_ASSERT`) for type and value assumptions, and runtime assertions (`ASSERT`) for precondition and invariant sanity checking. Do not use runtime assertions to check for errors as they should never alter control flow and potentially be excluded.
- Assume `UINT32/INT32` to be `int`-sized and use `%u`, `%d`, and `%x` to print them.
- Assume `UINTN/INTN` to be of unspecified size, and cast them to `UINT64/INT64` for printing with `%Lu`, `%Ld` and so on as normal.
- Do not rely on integer promotions for numeric literals. Use explicit casts when the type is implementation-dependent or suffixes when type size is known. Assume `U` for `UINT32` and `ULL` for `UINT64`.
- Do ensure unsigned arithmetics especially in bitwise maths, shifts in particular.
- `sizeof` operator should take variables instead of types where possible to be error prone. Use `ARRAY_SIZE` to obtain array size in elements. Use `L_STR_LEN` and `L_STR_SIZE` macros from `OcStringLib` to obtain string literal sizes to ensure compiler optimisation.
- Do not use `goto` keyword. Prefer early `return`, `break`, or `continue` after failing to pass error checking instead of nesting conditionals.
- Use `EFIAPI`, force UEFI calling convention, only in protocols, external callbacks between modules, and functions with variadic arguments.
- Provide inline documentation to every added function, at least describing its inputs, outputs, precondition, postcondition, and giving a brief description.
- Do not use `RETURN_STATUS`. Assume `EFI_STATUS` to be a matching superset that is to be always used when `BOOLEAN` is not enough.
- Security violations should halt the system or cause a forced reboot.

Codestyle. The codebase follows EDK II codestyle with few changes and clarifications.

- Write inline documentation for the functions and variables only once: in headers, where a header prototype is available, and inline for `static` variables and functions.
- Use line length of 120 characters or less, preferably 100 characters.
- Use spaces after casts, e.g. `(VOID *)(UINTN) Variable`.
- Use SPDX license headers as shown in acidanthera/bugtracker#483.

Debugging. The codebase incorporates EDK II debugging and few custom features to improve the experience.

- Use module prefixes, 2-5 letters followed by a colon (`:`), for debug messages. For `OpenCorePkg` use `OC:`, for libraries and drivers use their own unique prefixes.
- Do not use dots (`.`) in the end of debug messages and separate `EFI_STATUS`, printed by `%r`, with a hyphen (e.g. `OCRAM: Allocation of %u bytes failed - %r\n`).
- Use `DEBUG_CODE_BEGIN ()` and `DEBUG_CODE_END ()` constructions to guard debug checks that may potentially reduce the performance of release builds and are otherwise unnecessary.
- Use `DEBUG` macro to print debug messages during normal functioning, and `RUNTIME_DEBUG` for debugging after `EXIT_BOOT_SERVICES`.
- Use `DEBUG_VERBOSE` debug level to leave debug messages for future debugging of the code, which are currently not necessary. By default `DEBUG_VERBOSE` messages are ignored even in `DEBUG` builds.
- Use `DEBUG_INFO` debug level for all non critical messages (including errors) and `DEBUG_BULK_INFO` for extensive messages that should not appear in NVRAM log that is heavily limited in size. These messages are ignored in `RELEASE` builds.
- Use `DEBUG_ERROR` to print critical human visible messages that may potentially halt the boot process, and `DEBUG_WARN` for all other human visible errors, `RELEASE` builds included.

The addresses written here must be part of the memory map, have `EfiMemoryMappedIO` type and `EFI_MEMORY_RUNTIME` attribute (highest bit) set. To find the list of the candidates the debug log can be used.

2. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This address will be devirtualised unless set to `true`.

5.4 Quirks Properties

1. `AvoidRuntimeDefrag`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Protect from boot.efi runtime memory defragmentation.

   This option fixes UEFI runtime services (date, time, NVRAM, power control, etc.) support on many firmwares using SMM backing for select services like variable storage. SMM may try to access physical addresses, but they get moved by boot.efi.

   Note: Most but Apple and VMware firmwares need this quirk.

2. `DevirtualiseMmio`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Remove runtime attribute from select MMIO regions.

   This option reduces stolen memory footprint from the memory map by removing runtime bit for known memory regions. This quirk may result in the increase of KASLR slides available, but is not necessarily compatible with the target board without additional measures. In general this frees from 64 to 256 megabytes of memory (present in the debug log), and on some platforms it is the only way to boot macOS, which otherwise fails with allocation error at bootloader stage.

   This option is generally useful on all firmwares except some very old ones, like Sandy Bridge. On select firmwares it may require a list of exceptional addresses that still need to get their virtual addresses for proper NVRAM and hibernation functioning. Use `MmioWhitelist` section to do this.

3. `DisableSingleUser`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disable single user mode.

   This is a security option allowing one to restrict single user mode usage by ignoring `CMD+S` hotkey and `-s` boot argument. The behaviour with this quirk enabled is supposed to match T2-based model behaviour. Read this article to understand how to use single user mode with this quirk enabled.

4. `DisableVariableWrite`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Protect from macOS NVRAM write access.

   This is a security option allowing one to restrict NVRAM access in macOS. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`.

   Note: This quirk can also be used as an ugly workaround to buggy UEFI runtime services implementations that fail to write variables to NVRAM and break the rest of the operating system.

5. `DiscardHibernateMap`
   Type: `plist boolean`

Failsafe: `false`
Description: Reuse original hibernate memory map.

This option forces XNU kernel to ignore newly supplied memory map and assume that it did not change after waking from hibernation. This behaviour is required to work by Windows, which mandates to preserve runtime memory size and location after S4 wake.

Note: This may be used to workaround buggy memory maps on older hardware, and is now considered rare legacy. Examples of such hardware are Ivy Bridge laptops with Insyde firmware, like Acer V3-571G. Do not use this unless you fully understand the consequences.

6. `EnableSafeModeSlide`
Type: `plist boolean`
Failsafe: `false`
Description: Patch bootloader to have KASLR enabled in safe mode.

This option is relevant to the users that have issues booting to safe mode (e.g. by holding `shift` or using `-x` boot argument). By default safe mode forces `0` slide as if the system was launched with `slide=0` boot argument. This quirk tries to patch `boot.efi` to lift that limitation and let some other value (from `1` to `255`) be used. This quirk requires `ProvideCustomSlide` to be enabled.

Note: The necessity of this quirk is determined by safe mode availability. If booting to safe mode fails, this option can be tried to be enabled.

7. `EnableWriteUnprotector`
Type: `plist boolean`
Failsafe: `false`
Description: Permit write access to UEFI runtime services code.

This option bypasses `RX̂` permissions in code pages of UEFI runtime services by removing write protection (`WP`) bit from `CR0` register during their execution. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`.

Note: The necessity of this quirk is determined by early boot crashes of the firmware.

8. `ForceExitBootServices`
Type: `plist boolean`
Failsafe: `false`
Description: Retry `ExitBootServices` with new memory map on failure.

Try to ensure that `ExitBootServices` call succeeds even with outdated MemoryMap key argument by obtaining current memory map and retrying `ExitBootServices` call.

Note: The necessity of this quirk is determined by early boot crashes of the firmware. Do not use this unless you fully understand the consequences.

9. `ProtectCsmRegion`
Type: `plist boolean`
Failsafe: `false`
Description: Protect CSM region areas from relocation.

Ensure that CSM memory regions are marked as ACPI NVS to prevent boot.efi or XNU from relocating or using them.

Note: The necessity of this quirk is determined by artifacts and sleep wake issues. As `AvoidRuntimeDefrag` resolves a similar problem, no known firmwares should need this quirk. Do not use this unless you fully understand the consequences.

10. `ProvideCustomSlide`
Type: `plist boolean`
Failsafe: `false`
Description: Provide custom KASLR slide on low memory.

This option performs memory map analysis of your firmware and checks whether all slides (from `1` to `255`) can be used. As `boot.efi` generates this value randomly with `rdrand` or pseudo randomly `rdtsc`, there is a chance of

Failsafe: Empty string
Description: Kext executable path relative to bundle (e.g. `Contents/MacOS/Lilu`).

5. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Adds kernel driver on specified macOS version or older.

   Kernel version can be obtained with `uname -r` command, and should look like 3 numbers separated by dots, for example `18.7.0` is the kernel version for `10.14.6`. Kernel version interpretation is implemented as follows:

$$\begin{aligned} ParseDarwinVersion(\kappa, \lambda, \mu) = \kappa \cdot 10000 & \quad \text{Where } \kappa \in (0, 99) \text{ is kernel version major} \\ + \lambda \cdot 100 & \quad \text{Where } \lambda \in (0, 99) \text{ is kernel version minor} \\ + \mu & \quad \text{Where } \mu \in (0, 99) \text{ is kernel version patch} \end{aligned}$$

   Kernel version comparison is implemented as follows:

$$\alpha = \begin{cases} ParseDarwinVersion(\texttt{MinKernel}), & \text{If } \texttt{MinKernel} \text{ is valid} \\ 0 & Otherwise \end{cases}$$

$$\beta = \begin{cases} ParseDarwinVersion(\texttt{MaxKernel}), & \text{If } \texttt{MaxKernel} \text{ is valid} \\ \infty & Otherwise \end{cases}$$

$$\gamma = \begin{cases} ParseDarwinVersion(FindDarwinVersion()), & \text{If valid } \texttt{"Darwin Kernel Version"} \text{ is found} \\ \infty & Otherwise \end{cases}$$

$$f(\alpha, \beta, \gamma) = \alpha \leq \gamma \leq \beta$$

   Here $ParseDarwinVersion$ argument is assumed to be 3 integers obtained by splitting Darwin kernel version string from left to right by the `.` symbol. $FindDarwinVersion$ function looks up Darwin kernel version by locating `"Darwin Kernel Version` $\kappa.\lambda.\mu$`"` string in the kernel image.

6. `MinKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Adds kernel driver on specified macOS version or newer.

   Note: Refer to `Add MaxKernel` description for matching logic.

7. `PlistPath`
   Type: `plist string`
   Failsafe: Empty string
   Description: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

7.4 Block Properties

1. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This kernel driver will not be blocked unless set to `true`.

3. `Identifier`
   Type: `plist string`
   Failsafe: Empty string
   Description: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).

3. `AppleXcpmExtraMsrs`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables multiple MSR access critical for select CPUs, which have no native XCPM support.

   This is normally used in conjunction with `Emulate` section on Haswell-E, Broadwell-E, Skylake-X, and similar CPUs. More details on the XCPM patches are outlined in acidanthera/bugtracker#365.

   Note: Additional not provided patches will be required for Ivy Bridge or Pentium CPUs. It is recommended to use `AppleIntelCpuPowerManagement.kext` for the former.

4. `CustomSMBIOSGuid`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

5. `DisableIoMapper`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables `IOMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

   Note: This option is a preferred alternative to dropping `DMAR` ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.

6. `ExternalDiskIcons`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

   Note: This option should avoided whenever possible. Modern firmwares usually have compatible AHCI controllers.

7. `LapicKernelPanic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables kernel panic on LAPIC interrupts.

8. `PanicNoKextDump`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

9. `PowerTimeoutKernelPanic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables kernel panic on setPowerState timeout.

   An additional security measure was added to macOS Catalina (10.15) causing kernel panic on power change timeout for Apple drivers. Sometimes it may cause issues on misconfigured hardware, notably digital audio, which sometimes fails to wake up. For debug kernels `setpowerstate_panic=0` boot argument should be used, which is otherwise equivalent to this quirk.

10. ~~`ThirdPartyTrim`~~`ThirdPartyDrives`
    Type: `plist boolean`
    Failsafe: `false`
    Description: ~~Patch~~ Apply vendor patches to IOAHCIBlockStorage.kext to ~~force TRIM command support on AHCI SSDs~~ enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

    Note: This option ~~should avoided whenever possible~~may be avoided on user preference. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`. Starting from 10.15 this utility creates `EnableTRIM` variable in `APPLE_BOOT_VARIABLE_GUID` namespace with `01 00 00 00` value.

11. `XhciPortLimit`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

    Note: This option should be avoided whenever possible. USB port limit is imposed by the amount of used bits in locationID format and there is no possible way to workaround this without heavy OS modification. The only valid solution is to limit the amount of used ports to 15 (discarding some). More details can be found on AppleLife.ru.

5. `RequireVault`
   Type: `plist boolean`
   Failsafe: `true`
   Description: Require `vault.plist` file present in `OC` directory.

   This file should contain SHA-256 hashes for all files used by OpenCore. Presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not happen unnoticed. To create this file automatically use `create_vault.sh` script.

   Regardless of the underlying filesystem, path name and case must match between `config.plist` and `vault.plist`.

   Note: `vault.plist` is tried to be read regardless of the value of this option, but setting it to `true` will ensure configuration sanity, and abort the boot process.

   The complete set of commands to:

   - Create `vault.plist`.
   - Create a new RSA key (always do this to avoid loading old configuration).
   - Embed RSA key into `OpenCore.efi`.
   - Create `vault.sig`.

   Can look as follows:

```
cd /Volumes/EFI/EFI/OC
/path/to/create_vault.sh .
/path/to/RsaTool -sign vault.plist vault.sig vault.pub
off=$(($(strings -a -t d OpenCore.efi | grep "=BEGIN OC VAULT=" | cut -f1 -d' ')+16))
dd of=OpenCore.efi if=vault.pub bs=1 seek=$off count=520 conv=notrunc
dd of=OpenCore.efi if=vault.pub bs=1 seek=$off count=528 conv=notrunc
rm vault.pub
```

   Note: While it may appear obvious, but you have to use an external method to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this you are recommended to at least enable UEFI SecureBoot with a custom certificate, and sign `OpenCore.efi` and `BOOTx64.efi` with your custom key. More details on customising secure boot on modern firmwares can be found in Taming UEFI SecureBoot paper (in Russian).

6. `ScanPolicy`
   Type: `plist integer`, 32 bit
   Failsafe: `0xF0103`
   Description: Define operating system detection policy.

   This value allows to prevent scanning (and booting) from untrusted source based on a bitmask (sum) of select flags. As it is not possible to reliably detect every file system or device type, this feature cannot be fully relied upon in open environments, and the additional measures are to be applied.

   Third party drivers may introduce additional security (and performance) measures following the provided scan policy. Scan policy is exposed in `scan-policy` variable of `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` GUID for UEFI Boot Services only.

   - `0x00000001` (bit `0`) — `OC_SCAN_FILE_SYSTEM_LOCK`, restricts scanning to only known file systems defined as a part of this policy. File system drivers may not be aware of this policy, and to avoid mounting of undesired file systems it is best not to load its driver. This bit does not affect dmg mounting, which may have any file system. Known file systems are prefixed with `OC_SCAN_ALLOW_FS_`.
   - `0x00000002` (bit `1`) — `OC_SCAN_DEVICE_LOCK`, restricts scanning to only known device types defined as a part of this policy. This is not always possible to detect protocol tunneling, so be aware that on some systems it may be possible for e.g. USB HDDs to be recognised as SATA. Cases like this must be reported. Known device types are prefixed with `OC_SCAN_ALLOW_DEVICE_`.
   - `0x00000100` (bit `8`) — `OC_SCAN_ALLOW_FS_APFS`, allows scanning of APFS file system.
   - `0x00000200` (bit `9`) — `OC_SCAN_ALLOW_FS_HFS`, allows scanning of HFS file system.
   - `0x00000400` (bit `10`) — `OC_SCAN_ALLOW_FS_ESP`, allows scanning of EFI System Partition file system.
   - `0x00000800` (bit `11`) — `OC_SCAN_ALLOW_FS_NTFS`, allows scanning of NTFS (Msft Basic Data) file system.
   - `0x00001000` (bit `12`) — `OC_SCAN_ALLOW_FS_EXT`, allows scanning of EXT (Linux Root) file system.
   - `0x00010000` (bit `16`) — `OC_SCAN_ALLOW_DEVICE_SATA`, allow scanning SATA devices.

11 UEFI

11.1 Introduction

UEFI (Unified Extensible Firmware Interface) is a specification that defines a software interface between an operating system and platform firmware. This section allows to load additional UEFI modules and/or apply tweaks for the onboard firmware. To inspect firmware contents, apply modifications and perform upgrades UEFITool and supplementary utilities can be used.

11.2 Properties

1. `ConnectDrivers`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform UEFI controller connection after driver loading. This option is useful for loading filesystem drivers, which usually follow UEFI driver model, and may not start by themselves. While effective, this option is not necessary with e.g. APFS loader driver, and may slightly slowdown the boot.

2. `Drivers`
   Type: `plist array`
   Failsafe: None
   Description: Load selected drivers from `OC/Drivers` directory.

   Designed to be filled with string filenames meant to be loaded as UEFI drivers. Depending on the firmware a different set of drivers may be required. Loading an incompatible driver may lead your system to unbootable state or even cause permanent firmware damage. Some of the known drivers include:

   - `ApfsDriverLoader` — APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares.
   - `FwRuntimeServices` — `OC_FIRMWARE_RUNTIME` protocol implementation that increases the security of OpenCore and Lilu by supporting read-only and write-only NVRAM variables. Some quirks, like `RequestBootVarRouting`, require this driver for proper function. Due to the nature of being a runtime driver, i.e. functioning in parallel with the target operating system, it cannot be implemented within OpenCore itself.
   - `EnhancedFatDxe` — FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process.
   - `NvmExpressDxe` — NVMe support driver from `MdeModulePkg`. This driver is included in most firmwares starting with Broadwell generation. For Haswell and earlier embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
   - `UsbKbDxe` — USB keyboard driver adding the support of `AppleKeyMapAggregator` protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin ~~`KeySypport`~~`KeySupport`, which may work better or worse depending on the firmware.
   - `VirtualSmc` — UEFI SMC driver, required for proper FileVault 2 functionality and potentially other macOS specifics. An alternative, named `SMCHelper`, is not compatible with `VirtualSmc` and OpenCore, which is unaware of its specific interfaces. In case `FakeSMC` kernel extension is used, manual NVRAM variable addition may be needed and `VirtualSmc` driver should still be used.
   - `VBoxHfs` — HFS file system driver with bless support. This driver is an alternative to a closed source `HFSPlus` driver commonly found in Apple firmwares. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit.
   - `XhciDxe` — XHCI USB controller support driver from `MdeModulePkg`. This driver is included in most firmwares starting with Sandy Bridge generation. For earlier firmwares or legacy systems it may be used to support external USB 3.0 PCI cards.

   To compile the drivers from UDK (EDK II) use the same command you do normally use for OpenCore compilation, but choose a corresponding package:

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
source edksetup.sh
```

```
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p FatPkg/FatPkg.dsc
build -a X64 -b RELEASE -t XCODE5 -p MdeModulePkg/MdeModulePkg.dsc
```

3. `Input`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual settings designed for input (keyboard and mouse) in Input Properties section below.

4. `Protocols`
   Type: `plist dict`
   Failsafe: None
   Description: Force builtin versions of select protocols described in Protocols Properties section below.

   Note: all protocol instances are installed prior to driver loading.

5. `Quirks`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual firmware quirks described in Quirks Properties section below.

11.3 Input Properties

1. `KeyForgetThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Remove key unless it was submitted during this timeout in milliseconds.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers only report key presses as interrupts and pressing and holding the key on the keyboard results in subsequent submissions of this key with some defined time interval. As a result we use a timeout to remove once pressed keys from the buffer once the timeout expires and no new submission of this key happened.

   This option allows to set this timeout based on your platform. The recommended value that works on the majority of the platforms is `5` milliseconds. For reference, holding one key on VMware will repeat it roughly every `2` milliseconds and the same value for APTIO V is `3-4` milliseconds. Thus it is possible to set a slightly lower value on faster platforms and slightly higher value on slower platforms for more responsive input.

2. `KeyMergeThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

   Similarly to `KeyForgetThreshold`, this option works around the sequential nature of key submission. To be able to recognise simultaneously pressed keys in the situation when all keys arrive sequentially, we are required to set a timeout within which we assume the keys were pressed together.

   Holding multiple keys results in reports every `2` and `1` milliseconds for VMware and APTIO V respectively. Pressing keys one after the other results in delays of at least `6` and `10` milliseconds for the same platforms. The recommended value for this option is `2` milliseconds, but it may be decreased for faster platforms and increased for slower.

3. `KeySupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable internal keyboard input translation to `AppleKeyMapAggregator` protocol.

   This option activates the internal keyboard interceptor driver, based on `AppleGenericInput` aka (`AptioIntputFix`), to fill `AppleKeyMapAggregator` database for input functioning. In case a separate driver is used, such as `UsbKbDxe`, this option should never be enabled.

large memory chunks, such as macOS DMG recovery entries. On unaffected boards it may cause boot failures, and thus strongly not recommended. For known issues refer to `acidanthera/bugtracker#449`.

2. `ClearScreenOnModeSwitch`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Some firmwares clear only part of screen when switching from graphics to text mode, leaving a fragment of previously drawn image visible. This option fills the entire graphics screen with black color before switching to text mode.

   Note: `ConsoleControl` should be set to `true` for this to work.

3. `ExitBootServicesDelay`
   Type: `plist integer`
   Failsafe: `0`
   Description: Adds delay in microseconds after `EXIT_BOOT_SERVICES` event.

   This is a very ugly quirk to circumvent "Still waiting for root device" message on select APTIO IV firmwares, namely ASUS Z87-Pro, when using FileVault 2 in particular. It seems that for some reason they execute code in parallel to `EXIT_BOOT_SERVICES`, which results in SATA controller being inaccessible from macOS. A better approach should be found in some future. Expect 3-5 seconds to be enough in case the quirk is needed.

4. `IgnoreInvalidFlexRatio`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Select firmwares, namely APTIO IV, may contain invalid values in `MSR_FLEX_RATIO` (`0x194`) MSR register. These values may cause macOS boot failure on Intel platforms.

   Note: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required.

5. `IgnoreTextInGraphics`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Select firmwares output text onscreen in both graphics and text mode. This is normally unexpected, because random text may appear over graphical images and cause UI corruption. Setting this option to `true` will discard all text output when console control is in mode different from `Text`.

   Note: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required. This option may hide onscreen error messages. `ConsoleControl` may need to be set to `true` for this to work.

6. `ReplaceTabWithSpace`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Some firmwares do not print tab characters or even everything that follows them, causing difficulties or inability to use the UEFI Shell builtin text editor to edit property lists and other documents. This option makes the console output spaces instead of tabs.

   Note: `ConsoleControl` may need to be set to `true` for this to work.

7. `ProvideConsoleGop`
   Type: `plist boolean`
   Failsafe: `false`
   Description: macOS bootloader requires GOP (Graphics Output Protocol) to be present on console handle. This option will install it if missing.

8. `ReconnectOnResChange`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reconnect console controllers after changing screen resolution.

   On some firmwares when screen resolution is changed via GOP, it is required to reconnect the controllers, which produce the console protocols (simple text out). Otherwise they will not produce text based on the new resolution.

Note: On several boards this logic may result in black screen when launching OpenCore from Shell and thus it is optional. In versions prior to 0.5.2 this option was mandatory and not configurable. Please do not use this unless required.

9. `ReleaseUsbOwnership`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Attempt to detach USB controller ownership from the firmware driver. While most firmwares manage to properly do that, or at least have an option for, select firmwares do not. As a result, operating system may freeze upon boot. Not recommended unless required.

10. `RequestBootVarFallback`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Request fallback of some `Boot` prefixed variables from `OC_VENDOR_VARIABLE_GUID` to `EFI_GLOBAL_VARIABLE_GUID`.

    This quirk requires `RequestBootVarRouting` to be enabled and therefore `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`.

    By redirecting `Boot` prefixed variables to a separate GUID namespace we achieve multiple goals:

    - Operating systems are jailed and only controlled by OpenCore boot environment to enhance security.
    - Operating systems do not mess with OpenCore boot priority, and guarantee fluent updates and hibernation wakes for cases that require reboots with OpenCore in the middle.
    - Potentially incompatible boot entries, such as macOS entries, are not deleted or anyhow corrupted.

    However, some firmwares do their own boot option scanning upon startup by checking file presence on the available disks. Quite often this scanning includes non-standard locations, such as Windows Bootloader paths. Normally it is not an issue, but some firmwares, ASUS firmwares on APTIO V in particular, have bugs. For them scanning is implemented improperly, and firmware preferences may get accidentally corrupted due to `BootOrder` entry duplication (each option will be added twice) making it impossible to boot without cleaning NVRAM.

    To trigger the bug one should have some valid boot options (e.g. OpenCore) and then install Windows with `RequestBootVarRouting` enabled. As Windows bootloader option will not be created by Windows installer, the firmware will attempt to create it itself, and then corrupt its boot option list.

    This quirk forwards all UEFI specification valid boot options, that are not related to macOS, to the firmware into `BootF###` and `BootOrder` variables upon write. As the entries are added to the end of `BootOrder`, this does not break boot priority, but ensures that the firmware does not try to append a new option on its own after Windows installation for instance.

11. `RequestBootVarRouting`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Request redirect of all `Boot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

    This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`. The quirk lets default boot entry preservation at times when firmwares delete incompatible boot entries. Simply said, you are required to enable this quirk to be able to reliably use Startup Disk preference pane in a firmware that is not compatible with macOS boot entries by design.

12. `SanitiseClearScreen`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Some firmwares reset screen resolution to a failsafe value (like `1024x768`) on the attempts to clear screen contents when large display (e.g. 2K or 4K) is used. This option attempts to apply a workaround.

    Note: `ConsoleControl` may need to be set to `true` for this to work. On all known affected systems `ConsoleMode` had to be set to empty string for this to work.

13. ~~`ClearScreenOnModeSwitch`~~`UnblockFsConnect`
    Type: `plist boolean`

Failsafe: `false`
Description: Some firmwares ~~clear only part of screen when switching from graphics to text mode, leaving a fragment of previously drawn image visible. This option fills the entire graphics screen with black color before switching to text mode~~block partition handles by opening them in By Driver mode, which results in File System protocols being unable to install.

Note: ~~`ConsoleControl` should be set to `true` for this to work~~The quirk is mostly relevant for select HP laptops with no drives listed.

- Logging is enabled (1) and shown onscreen (2): Misc → Debug → Target = 3.
- Logged messages from at least DEBUG_ERROR (0x80000000), DEBUG_WARN (0x00000002), and DEBUG_INFO (0x00000040) levels are visible onscreen: Misc → Debug → DisplayLevel = 0x80000042.
- Critical error messages, like DEBUG_ERROR, stop booting: Misc → Security → HaltLevel = 0x80000000.
- Watch Dog is disabled to prevent automatic reboot: Misc → Debug → DisableWatchDog = true.
- Boot Picker (entry selector) is enabled: Misc → Boot → ShowPicker = true.

If there is no obvious error, check the available hacks in Quirks sections one by one. For early boot troubleshooting, for instance, when OpenCore menu does not appear, using UEFI Shell may help to see early debug messages.

2. **How to customise boot entries?**

   OpenCore follows standard Apple Bless model and extracts the entry name from .contentDetails and .disk_label.contentDetails files in the booter directory if present. These files contain an ASCII string with an entry title, which may then be customised by the user.

3. **How to choose the default boot entry?**

   OpenCore uses the primary UEFI boot option to select the default entry. This choice can be altered from UEFI Setup, with the macOS Startup Disk preference, or the Windows Boot Camp Control Panel. Since choosing OpenCore's BOOTx64.EFI as a primary boot option limits this functionality in addition to several firmwares deleting incompatible boot options, potentially including those created by macOS, you are strongly encouraged to use the RequestBootVarRouting quirk, which will preserve your selection made in the operating system within the OpenCore variable space. Note, that RequestBootVarRouting requires a separate driver for functioning.

4. **What is the simplest way to install macOS?**

   Copy online recovery image (*.dmg and *.chunklist files) to com.apple.recovery.boot directory on a FAT32 partition with OpenCore. Load OpenCore Boot Picker and choose the entry, it will have a (dmg) suffix. Custom name may be created by providing .contentDetails file.

   To download recovery online you may use macrecovery.py tool from MacInfoPkg.

   For offline installation refer to How to create a bootable installer for macOS article.

5. **Why do online recovery images (*.dmg) fail to load?**

   This may be caused by missing HFS+ driver, as all presently known recovery volumes have HFS+ filesystem. Another cause may be buggy firmware allocator, which can be worked around with AvoidHighAlloc UEFI quirk.

6. **Can I use this on Apple hardware or virtual machines?**

   Sure, most relatively modern Mac models including MacPro5,1 and virtual machines are fully supported. Even though there are little to none specific details relevant to Mac hardware, some ongoing instructions can be found in acidanthera/bugtracker#377.

7. **Why do Find&Replace patches must equal in length?**

   For machine code (x86 code) it is not possible to do ~~such~~ differently sized replacements due to relative addressing. For ACPI code this is risky, and is technically equivalent to ACPI table replacement, thus not implemented. More detailed explanation can be found on AppleLife.ru.

8. **How can I migrate from AptioMemoryFix?**

   Behaviour similar to that of AptioMemoryFix can be obtained by installing FwRuntimeServices driver and enabling the quirks listed below. Please note, that most of these are not necessary to be enabled. Refer to their individual descriptions in this document for more details.

   - ProvideConsoleGop (UEFI quirk)
   - AvoidRuntimeDefrag
   - DiscardHibernateMap
   - EnableSafeModeSlide
   - EnableWriteUnprotector
   - ForceExitBootServices
   - ProtectCsmRegion
   - ProvideCustomSlide